OccuRE: an Occupancy REasoning Platform for Occupancy-driven Applications

Mikkel Baun Kjærgaard, Aslak Johansen
Fisayo Sangogboye, Emil Holmegaard
SDU Center for Energy Informatics
Mærsk McKinney Møller Institute
University of Southern Denmark
Email: {mbkj,asjo,fsa,em}@mmmi.sdu.dk

***Abstract*—Occupant behavior determines a large share of the energy consumption of buildings. Software applications driven by information about occupant behavior provide a mean to optimize this share. However, existing systems for sensing occupancy behavior provide technology-specific APIs statically coupled to the type of computed occupancy information. Software platforms for developing applications for buildings do also not provide abstractions for occupancy behavior. Therefore, technology lock in and lack of proper abstractions wreck the development of occupancy-driven applications. In this paper we present the design, implementation and evaluation of OccuRE, a stream-based Occupancy REasoning platform. OccuRE provides a technology agnostic API for accessing occupancy information to significantly improve portability. The platform uses a component-based computation model with dynamic composition to calculate and reason about occupancy behavior. Together these elements avoid that developers need to deal with technology-specific processing of sensor data to ease application development. Through micro-benchmarks we show that OccuRE successfully and efficiently computes occupancy information for technology-heterogeneous building instrumentations. We use the development of three prototype applications to demonstrate that the API of OccuRE (i) enables several types of occupancy-driven applications, (ii) that the applications – by using the interface – achieve portability in regards to occupancy information computation and (iii) that the application code avoids handling sensor data processing.**

I. Introduction

To create a sustainable society it is important to improve the energy performance of buildings. Especially the impact of occupant behavior on energy consumption is an important challenge [1]. Occupant behavior here refers to *all actions of the occupant (including presence) that affect building energy consumption* [2]. Occupancy information that quantitatively describes occupant behavior enables software-based decision logic that reacts on occurring behavior. Such software applications could improve the intelligence of equipment or infrastructures to better adapt to occupancy behavior, thus providing a slimmer fit to the (ideal) utility and comfort requirements. They could even go one step further by involving occupants in order to change their behaviors to be more energy efficient.

The concept of software-defined buildings [3] aims to improve the operation of buildings by providing an information platform for the creation of efficient and human-centered building systems. A core challenge within this vision is the creation of portable software applications to scale deployments to large parts of the building stock. Portability following the ISO 9126-1 software quality model refers to how well software can adapt to changes in its environment or requirements. Particular relevant in a building setting is the environment (i.e. building instrumentation). To enable portable software applications for buildings the community has proposed several types of Building Operating Systems (BOS) sandboxing applications from the particular instrumentation of a building. These software platforms include research driven efforts (e.g., sMAP [4], BOSS [3] and BuildingDepot [5]), open source community projects (e.g. OpenHAB [6]) and industry driven efforts (e.g., HomeOS [7]). The BOS platforms improve portability of applications at the level of the building instrumentation.

Given a BOS platform an application seeking occupancy information can interface in each building with available occupancy sensing systems. However, existing occupancy sensing systems (e.g. [8], [9], [10], [11]) provide technology-specific Application Programming Interfaces (APIs) and data streams that statically couple individual sensors to the type of computed occupancy information. Additionally the systems often fix the temporal and spatial granularity of information and a narrow temporal coverage. To address these shortcomings BOS platforms have to be extended with new abstractions to overcome these issues to increase portability and ease of development for occupancy-driven applications.

In this paper we present OccuRE, a stream-based Occupancy REasoning platform for occupancy information. The platform extends BOS platform functionality with a component-based computation model for occupancy information and a technology agnostic API for occupancy information. These extensions enable the development of portable occupancy-driven applications and ease application development. The contributions of the paper are as follows:

- **An API for occupancy information.** The API enables highly portable applications by providing technology agnostic abstractions with customizable temporal and spatial granularity and coverage. The API design was informed by a literature review of previous work on occupancy-driven applications.
- **A computation model for occupancy information** that componentize processing into strategies that consists of a series of reusable steps. The strategies are then

dynamically selected at runtime to calculate requested information. The information is provided in the best possible accuracy without a strict dependency on the actual building instrumentation.

- **The OccuRE stream-based Occupancy REasoning platform** which implements the API and computation model. The implementation provides easy means for specifying API requests as building stream configurations. For computing occupancy information, OccuRE currently provides 22 processing strategies with reusable processing steps that cover a variety of scenarios. Additionally, the platform provides a range of OccuRE-enabled building instrumentation drivers to minimize the effort of adding new types of occupancy sensing systems.
- **Evaluation results** based on micro-benchmarks and prototyping of occupancy-driven applications. The micro-benchmarks support that OccuRE successfully and efficiently computes a variety of occupancy information. The results cover the technology-heterogeneous instrumentation of three different buildings. The evaluation of three application prototypes demonstrates that the API of OccuRE enables key types of occupancy-driven applications. Additionally, that the applications by using the interface achieve building portability in regards to occupancy information computation. Finally, that OccuRE eases application development by avoiding application code for complex sensor data processing needed when using an existing BOS platform.

OccuRE is envisioned to support occupancy-driven applications, such as, personal building controls, predictive occupancy-driven heating, activity-based equipment control and building diagnostics. Applications also exist in other domains than energy performance including building safety and maintenance. This paper reports the experiences of developing three energy performance applications on top of the platform. Thereby, the platform is an important step towards the development of portable occupancy-driven applications to improve the energy performance of buildings.

II. System Design Goals

For informing the design of the API and computation model, we surveyed more than forty articles from a broad selection of relevant venues. The articles cover occupancy-driven applications, sensor modalities for occupancy sensing and processing strategies for computing occupancy information. From the survey we distill three design goals for OccuRE.

From the surveyed literature, five categories of domain aspects were identified to be important for occupancy-driven applications: type of occupancy information, temporal granularity, spatial granularity, temporal coverage and spatial coverage. Occupancy information type covers the different ways occupants can impact the energy consumption of a building; including presence, individual actions or behavior. Temporal coverage includes the subcategories: *past* denoting that the application uses occupancy information about the past, *now* denoting that the application uses occupancy information

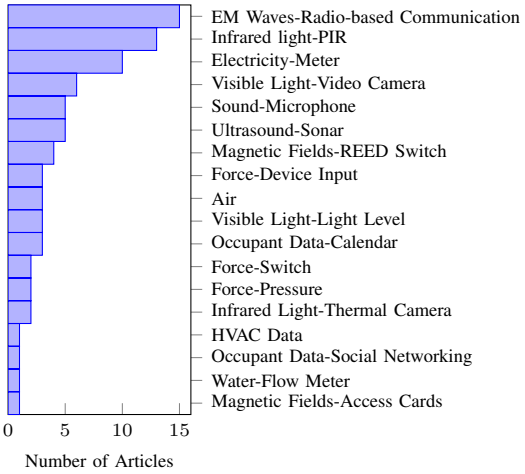

Fig. 1. Identified sensor modalities from previous work.

for the most recent point in time and *future* denoting that the system uses occupancy information predicted for the future. Temporal granularity describes the temporal resolution of occupancy information. Spatial granularity and coverage characterize the spatial resolution and extent of information, respectively. In our work we use values following the widely recognized IFC standard, such as, *building* a physical structure placed on a site, *space* a subpart of a building which might correspond to a HVAC zone, a room or a subpart of a room, or a cubicle, and *object* a physical element placed in a space, e.g., a desk, chair or coffee machine.

Existing work on occupancy-driven applications were categorized according to these domain categories. Table I shows the categorization of different applications in regards to temporal coverage, spatial granularity and type of information. From the table we make the following observations: 1) Multiple applications for buildings exists, which use all types of occupancy information, at a different temporal and spatial coverage and granularity. 2) The same type of building application has been constructed based on information at a different temporal and spatial coverage. These observations inform **Design Goal - DG1:** *Provide an API for computing every type of occupancy information with different temporal and spatial coverage and granularities. The API should sandbox applications from the building instrumentation to increase building portability.*

We also surveyed the sensor modalities used by existing occupancy sensing systems. Figure 1 shows the 18 identified sensor modalities. Furthermore, it was observed that the sensing systems were implemented with different types of hardware, software and data access interfaces. In terms of data processing the systems covered the spectrum from providing unprocessed data to highly processed data. This leads to **Design Goal - DG2:** *Support the integration in OccuRE of building instrumentations deploying a variety of occupancy sensing systems. Additionally, provide means for representing relevant temporal and spatial metadata about the occupancy sensing systems to enable further processing.*

Finally, we surveyed the data processing strategies applied by different occupancy sensing systems. Figure 2 shows the

	Building	Spaces (E.g. HVAC Zone, Room, Cubicle)	Object
Past	Energy Use vs. Occupant Counts [8], [12]	Building Performance Visualisation [13], Simulation of Occupant Driven Building Control [9], Occupant Schedule Mining for Building Simulation [14], [15], [16], Disaggregation of Occupant Energy Usage [17]	Disaggregation of Occupant Energy Usage [18], [17]
Now	Personal Building Controls [19], Occupant Presence Driven Heating [20], [11]	Occupant Presence Driven HVAC [21], Occupant Count Driven Ventilation [22], [19], Presence-based Equipment Control [8], [12], Activity-based Equipment Control [23], [24]	Activity-based Equipment Control [23], [24]
Future	Predicted Occupant Presence Driven Heating [20], [11]	Predicted Occupant Presence Driven Heating [11], Predicted Occupant Count Driven HVAC [25], [26], [10], [27]	*– No Examples Found –*

TABLE I
CATEGORIZATION OF APPLICATIONS FOR BUILDINGS REPORTED BY EXISTING WORK.

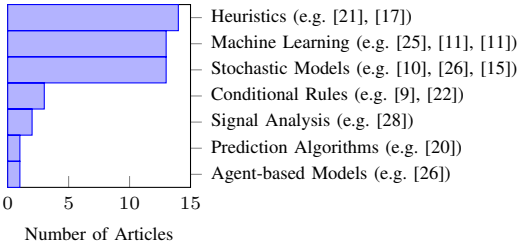

Fig. 2. Identified processing strategies in previous work.

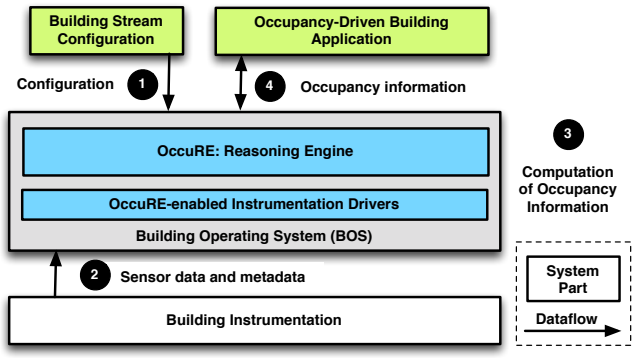

Fig. 3. Overview of OccuRE. The platform includes building stream configurations, OccuRE-enabled building instrumentation drivers and the OccuRE reasoning engine for occupancy information computation.

result with the individual strategies categorized into seven broad types of methods. The result illustrates the breath of methods applicable to the processing of sensor data to compute occupancy information. This motivates **Design Goal - DG3:** *Provide an abstraction that enables developers to provide implementations for a broad selection of sensor data processing methods for occupancy sensing.*

III. OccuRE at a Glance

Figure 3 highlights the high-level architecture and the flow of information. The design of OccuRE is defined on top of a typical BOS platform. The BOS is assumed to provide integration capabilities for heterogeneous sensors and actuators, a communication layer for streaming of sensor data, storage of historical sensor data, and access control. The sMAP platform [4] was used for concretization. The sMAP system provides the listed BOS features based on REST interfaces, a query language for metadata and sensor data, and the ReadingDB time-series database.

The flow of information in OccuRE is sequenced in Figure 3: (1) A building stream configuration encapsulate a API request for OccuRE to compute occupant information streams. (2) OccuRE-enabled building instrumentation drivers collect

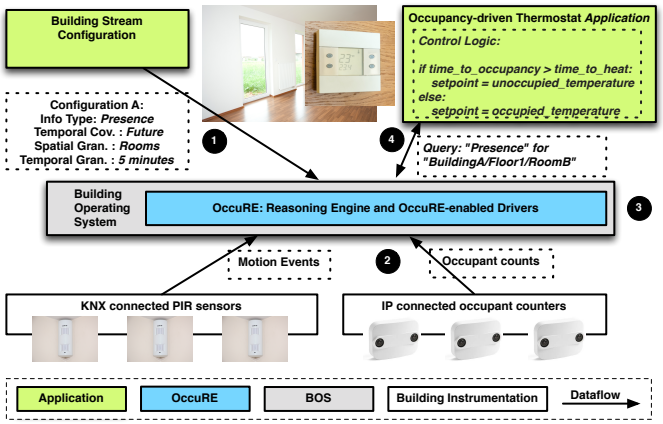

Fig. 4. Occupancy-driven thermostat application using OccuRE to compute predictions for the time to occupancy for each room.

data from occupancy sensing systems and provide needed temporal and spatial metadata. OccuRE resolves which sensor data streams to use for computing requested information and via the BOS collects sensor data from the selected data streams. (3) The OccuRE reasoning engine selects a processsing strategy and applies it to collected data. The engine afterwards streams the computed occupancy information back to the BOS. (4) At a later point an application subscribes via queries to receive data from one or more streams.

As a concrete example Figure 4 shows the realization of a occupancy-driven thermostat application [11] on top of OccuRE. In general, rooms only need to keep a comfortable condition (temperature) when they are occupied. When unoccupied, energy can be saved by allowing the temperature to drift away from the comfort temperature. However, a space conditioning system takes time to heat (or cool) a room to reach a comfort temperature. Therefore predictions of the time to occupancy can help determining when to heat a room in order to reach the prescribed comfort temperature at a minimum of wasted energy.

For the application the flow of information is sequenced in Figure 4. (1) The occupancy-driven thermostat application provides a building stream configuration for the API that requests OccuRE to compute presence in the future for controlled rooms. (2) OccuRE-enabled building instrumentation drivers interact with the building instrumentation through the BOS to provide streams of sensor data. (3) OccuRE initiates the prediction of presence in the future for each room based on available streams of sensor data. (4) The occupancy-driven thermostat application queries computed information and runs

it's control logic to switch between a comfort temperature when the room is soon to be occupied and a disabled/floating setpoint when the room is unlikely to be occupied. By using the OccuRE API the thermostat application is decoupled from both the building instrumentation and the computation of occupancy information. These independences each provide a degree of portability and ease application development.

IV. System Abstractions

To realize the listed design goals OccuRE exposes the following abstractions to application developers.

Streams. The streams provide time-series data following the sMAP design [4] representing sensor readings or processed occupancy information. The advantage of the stream abstraction is that it naturally supports online processing of data required in most application scenarios. The frequency of data depends on the temporal granularity specified either by a building stream configuration or by the frequency of data provided by the building instrumentation. Streams can be sent across a network. The streams are self-descriptive and the sMAP platform provides means to query metadata about available streams. Information included in the metadata covers the data source, type of data, and spatial and temporal properties. Applications can subscribe to streams with the extensive sMAP query language [4] that enable selection, projection and aggregation over metadata and streaming data.

Building Stream Configurations. The configurations encapsulate API requests to compute different kinds of occupancy information, coverage and granularity. The benefit of configurations is that they decouple the request for occupancy information from the data processing implementation. The computed streams of occupancy information can either be configured per building or established ad-hoc on request by applications. The building stream configurations specify temporal coverage (e.g. `future T`, `now` or `past T` where `T` specify the time period), temporal granularity (e.g. `1Hour` or `available` to use the granularity of the sensor data), spatial coverage (e.g. extent of `building X` or `space Y`), spatial granularity (e.g. resolution of `building` or `space`), occupancy information type (e.g. `presence-boolean`, `presence-count` or `office-activity-model` with values, such as, `at-desk`, `in-meeting` and `away`) and invocation model (e.g. `event-based` or `periodically-5Min`) to specify when information is computed. Thereby, they provide the functionality to achieve design goal DG1.

OccuRE-enabled Building Instrumentation Drivers. The drivers collect data from occupancy sensors and provide temporal and spatial metadata about occupancy sensors. The driver's role is to decouple a particular device in a building instrumentation from the processing of its sensor data. The decoupling enables the construction of a rich set of reusable drivers for different types of building instrumentations. These drivers are either built from scratch or by upgrading existing sMAP drivers that already implement communication integration to the sensors of a building instrumentation. The metadata covers the spatial placement of sensors (i.e., the site, building, building story or space of installation), sensor type, sensor quality, sensor data form and if necessary, additional placement information. Placement information might be in relation to a model or coordinate system or denote the coverage angle of a sensor. The drivers are configured using the configuration file concept developed for sMAP [4]. Thereby, they provide the functionality to achieve design goal DG2.

Drivers	Description	Sensor Modalities
KnxViaOPC	Access KNX building instrumentation via an OPC server. Metadata extracted from OPC item descriptions	PIR, Light switches, CO_2
RazBerry	Acesss Z-wave devices via a Raspberry with a Razberry board. Metadata extracted from Z-wave device descriptions	PIR, Power switch, CO_2, REED switches
XovisCounter	Access Xovis camera counters via web API. Metadata extacted from count item descriptions	camera count lines
ODataWebAPI	Access web API based on OData	PIR, CO_2
SmapWrapper	Replicate any sMAP stream and add metadata	Any type

TABLE II
Metadata instrumentation drivers.

To provide a rich set of drivers for the OccuRE platform both new drivers were implemented and existing sMAP instrumentation drivers were upgraded to OccuRE-enabled building instrumentation drivers. Also implemented was a wrapper driver that can attach metadata to existing data streams of an sMAP server making it straight forward to use any existing running sMAP server, e.g., openbms.org. Table II lists the implemented set of drivers.

Processing Strategies. The strategies from streams of sensor data compute occupancy information. Each processing strategy has to implement the following steps: 1) specify supported sensor modalities and computed occupancy information; 2) estimate the quality of information that the strategy can deliver given available streams of sensor data; 3) select relevant sensor data streams for computing occupancy information for a particular spatial granularity and coverage; 4) query data from each individual stream to compute information with the requested temporal coverage; 5) as an optional step learn a model from data for the particular setting. Learned models are cached by OccuRE between each invocation; 6) compute occupancy information from sensor data. The individual strategies might implement methods for predictions, data mining of patterns, fusion of different data types, mapping among data types and scaling and resampling of temporal and spatial granularities. The individual steps are defined and exposed through an interface. Therefore, developers can reuse or add new processing strategies or steps if existing ones does not cover a particular information type or sensor modality.

As an example a processing strategy has the goal to compute an occupancy presence count with temporal coverage of now and a spatial coverage of a building. A developer decides that a strategy will use the amount of triggered PIR sensors to make a rough estimate of the real occupancy count. Hence, this strategy will provide the following implementations of

the six steps as also shown in Listing 1 and explained in the following. 1) specify the used sensor modality as `PIR` and the provided occupancy information as presence `COUNT` for the `NOW`. For 2) a method that estimates the quality of information based on the amount of rooms covered with PIR sensors, 3) a method that groups these per building, 4) a method that queries the most recent reading for all PIR sensors within the coverage area and 5) as no model is learned the step will not be implemented. For 6) a method that counts the number of triggered PIR sensors and return the count as the result. In this example we only return one time stamped value. When computing info for the past or the future the strategies will return a time-series for the relevant time period.

Listing 1. for computing counts from PIR sensors

```
class CountNowPirStrategy(OccuREStrategy):

    def fromData(self): # Step 1
        return [OccupancyTerms.PIR]

    def toData(self): # Step 1
        return [OccupancyTerms.COUNT,OccupancyTerms.NOW]

    def estimateQuality(self,spatialareas,datapoints):  #
         Step 2
        sensorquality = 0
        for datapoint in datapoints:
            sensorquality = sensorquality + datapoint['
                 quality']
        return sensorquality/len(spatialareas)

    def dataGroupingStrategy(self): # Step 3
        return BySpatialGranularity()

    def dataSelectionStrategy(self): # Step 4
        return DataSelectionNow()

    def apply(self,spatialkey,metadata,data,resampler,
         request): # Step 6
        result = 0
        for entry in data:
            if entry['Readings'][0] == 0:
                result = result + 1
        return result
```

The platform implementation currently provides 22 different strategies as listed in Table III. The strategies enable the computation of several types of occupancy information from different types of sensors. The strategies include both rather simple ones as given in the example above and more advanced ones building on previous work [29], [9]. Due to space restrictions we have to omit details on the individual strategies. The platform also supports the development of new strategies by allowing developers to implement the `OccuREStrategy` interface and register the strategy with the platform service. The developer might also include code to request external services if relevant, e.g., to improve occupancy prediction one might query upcoming public holidays from a web service. A developer might implement the steps from scratch or utilize the rich set of step implementations available with the platform. For instance, there are two implementations used by the currently available strategies for step 3. The first implementation groups sensors automatically by their metadata according to the requested spatial granularity and is used by 6 strategies. E.g., group PIR sensors by the building floor they belong to (if the spatial granularity is building floors).

	PIR	REED	CC Lines	Past	Now	Future	Presence	Count	Strategies
Boolean operator on motion events	X			X	X	X	X		3
Sum motion events	X			X	X	X		X	3
Sum count line crossings			X	X	X	X		X	3
Positive count line crossings			X	X	X	X	X		3
Divide counts with motion events	X		X	X	X	X		X	3
Fuse door opening and motion events	X	X		X	X	X	X		3
Multi-label Classification	X					X	X		1
Multi-label Regression			X			X		X	1
Commonality Profiles using Clustering	X	X	X	X			X	X	2

TABLE III
PROPERTIES OF THE 22 STRATEGIES CURRENTLY AVAILABLE.

The second implementation assumes that sensors need to be mapped by a model from one spatial granularity into another spatial granularity and is used by 12 strategies. An example is the counting lines monitored by camera-based occupancy sensors. To produce total occupancy counts for an area the counts has to be summed for the individual camera counting lines. This is achieved with a spatial model that specify the areas that a set of lines create or for a specific area the set of counting lines that delimits the area.

V. SYSTEM COMPONENTS

Figure 5 shows the system components of the OccuRE platform and utilized system components of sMAP. The OccuRE system components have been designed to combine the introduced abstractions as follows. (1) The OccuRE reasoning engine invokes a *SensorDataResolver* to resolve what streams of sensor data are available and relevant for satisfying a given building stream configuration by querying the sMAP `Metadata Broker`. The *SensorDataResolver* uses the spatial metadata to determine the relevance of each stream. (2) Given the set of relevant streams of sensor data, a *StrategyResolver* selects the best processing *Strategy*. (3) The chosen `Strategy` is scheduled to produce the requested occupancy information based on requested data from the sMAP `Stream Archiver`. (4) The processing steps might apply methods for `Temporal and Spatial Resampling` to either downsample or upsample to provide the requested granularity before the computed data is streamed back to the sMAP `Stream Archiver`. Occupancy information might be privacy sensitive information. Therefore, the sMAP access control scheme is applied to regulate access to the streams of occupancy information using the `Access Control` componet of sMAP.

SensorDataResolver. The responsibility of the SensorDataResolver is to identify available streams of sensor data which can aid in serving a building stream configuration. The component thereby decouples the processing strategy implementations from the identification of sensor data. The identification process considers both the spatial coverage area and any explicitly requested sensor modalities. The *SensorDataResolver* will query metadata from the `Metadata`

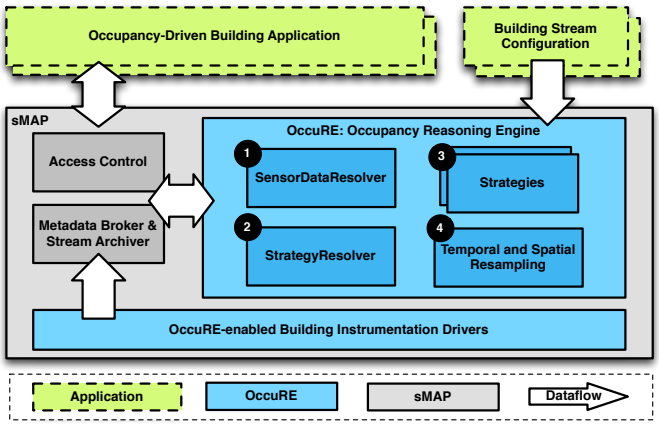

Fig. 5. OccuRE system architecture covering stream configurations, streams, processing strategies and instrumentation drivers.

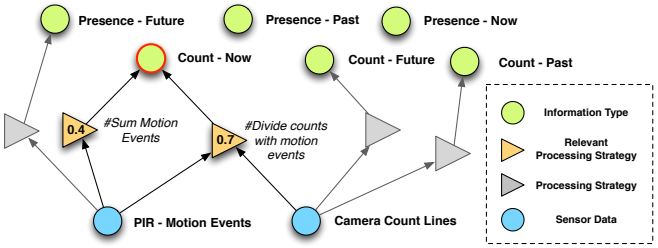

Fig. 6. *StrategyResolver* representation of sensor data, processing strategies and occupancy information for a building stream configuration to produce count information for the now.

`broker` about available sensor data streams to identify relevant streams. From the requested metadata the *SensorDataResolver* will build up a list of potential sensor data sources including metadata that can be used to compute requested information. The list is recomputed continuously to handle if sensors apear or disapear.

StrategyResolver. The responsibility of the *StrategyResolver* is to select the best processing strategy for computing the requested occupancy information. The component thereby decouples the processing strategy implementations from the building stream configuration processing.

The OccuRE system registers at startup all available processing strategies with the *StrategyResolver.* The *StrategyResolver* represents each strategy in a directed graph that links types of sensor data to occupancy information as shown in Figure 6. Given a building stream configuration the *StrategyResolver* analyses available sensor data and identify applicable processing strategies. Afterwards it asks each strategy to estimate what accuracy it can deliver (illustrated as numbers in the figure). The *StrategyResolver* then finally selects and initiates the strategy that delivers the highest quality of information. It might happen that none or not enough sensors are available in the coverage area to satisfy the requirements of any strategies to compute the requested information. In such cases the platform provides an error message with information about unsatisfied constraints. The error information enables developers and platform managers to either increase available sensors or implement a new strategy that can make do with the available sensor data.

VI. Implementation

The OccuRE platform is implemented in Python as an extension of the sMAP platform with the described abstractions and system components. The OccuRE platform is implemented on top of the Twisted event-driven networking engine. The OccuRE platform loads building stream configurations and based on these it allocates threads for processing sensor data for occupancy information. For internal representation of time-series the platform utilize the pandas framework for time-series. For graph processing the `StrategyResolver` uses the networkx graph framework. The sMAP access control is based on API keys. A building stream configuration has to provide an API key that determines the set of streams that the platform will have access to. The API key also determines the access rights for the streams of occupancy information that OccuRE publishes via sMAP. Applications can then subscribe to streams via the sMAP query system based on relevant metadata using either the sMAP client library written in Python or a REST style web API. For instance, to get a stream providing the total count of occupants in a building an application would issue the following query:

```
select data before now limit 1 where
Metadata/Building='M' and Metadata/Form =
'OccupancyCount'
```

VII. Evaluation

In this section the performance of the OccuRE platform is evaluated with micro-benchmarks. The goal is to demonstrate that the OccuRE platform has a small memory footprint, good runtime performance, correctly computes occupancy information, sandbox applications to increase building portability and is easily extended with new processing strategies. The micro-benchmarks consider the technology-heterogeneous building instrumentation of three different buildings.

For the evaluation we deployed the OccuRE platform on a virtual machine with four CPUs with a hypervisor from Microsoft and sMAP on another virtual machine with two CPUs with a hypervisor from VMware. The installed version of sMAP was the one available from github.org in May 2015. Both servers runs Ubuntu 12.04 and are connected by an internal network with an average ping time of 1.5 ms and a iperf measured bandwidth of 846 Mbits/sec. Using the Phoronix Test Suite v3.6.1 with the scimark2 benchmark, the servers were benchmarked to 1004 Mflops and 984 Mflops, respectively. The sMAP server is integrated with the building instrumentation of building M; a 2562 m^2 office building at the University of Southern Denmark. The sMap server receives sensor data streams via the KnxViaOPC driver from 68 PIR sensors and 68 light switches, the RazBerry driver for a PIR, a CO2, a door and power switch sensor installed in a single office, and the XovisCounter driver for 30 camera-based counting lines. To evaluate the system, seven representative building stream configurations were specified as listed in Table IV. The building stream configurations request OccuRE to compute occupancy information for the whole building and

	Now-PIR	Past-PIR	Future-PIR	Now-Counts	Now-Fusion	Now-Spaces	Now-1Hour
Information Type	Presence-Boolean	Presence-Boolean	Presence-Boolean	**Presence-count**	**Presence-count**	Presence-Boolean	Presence-Boolean
Temp. Coverage	**Now**	**Past-1Day**	**Future-1Day**	Now	Now	Now	Now
Temp. Granularity	5Min	5Min	5Min	5Min	5Min	5Min	**1Hour**
Spatial Granularity	Spaces	Spaces	Spaces	Spaces	Spaces	**Building**	Spaces
Sensor Modalities	PIR	PIR	PIR	**Counter**	**Counter+PIR**	PIR	PIR

TABLE IV
EVALUATION CONFIGURATIONS

	Files	LoC
Metadata Inst. Drivers	6	551
Processing Strategies and Steps	18	777
System Components	9	329
Total	33	1657

TABLE V
OCCURE PLATFORM - LINES OF CODE (LOC).

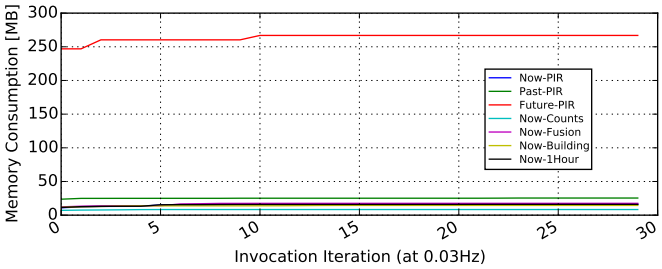

Fig. 7. Memory consumption of OccuRE for seven configurations.

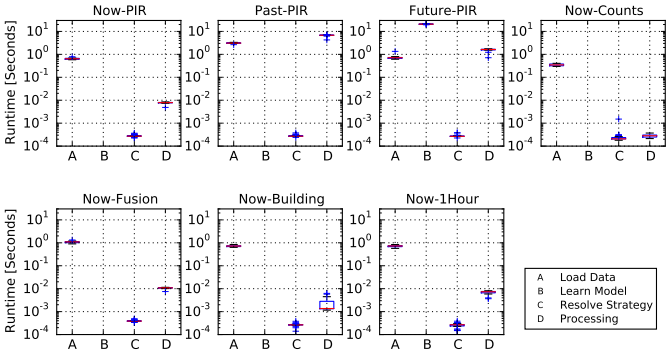

Fig. 8. Running time distribution for seven configurations.

utilizes the different features of the OccuRE API to reach all features of the platform.

Source Code and Memory Footprint. The OccuRE platform implementation consists of processing strategies, OccuRE-enabled building instrumentation drivers and system components. Table V lists the lines of code for each of these parts computed by the `cloc` tool. To evaluate the memory consumption used by the OccuRE platform during execution, the memory consumption of the platform was logged while executing the building stream configurations listed in Table IV. Figure 7 shows the results of executing each building stream configuration for 15 minutes with a 30 second delay between each invocation iteration. From the figure it can be observed that for most building stream configurations, the OccuRE platform loads and quickly converges to run using approximately 10 MB. The only exception is the strategy behind *Future-PIR* that learns a model for all spaces in building M and thereby uses up to 270 MB to hold the model in memory. The implementation of the strategy behind *Future-PIR* was made with no consideration for memory consumption. However, memory consumption can be significantly reduced by dumping the learned model to a database or compressing the model in both the temporal and spatial dimensions.

Runtime Performance. To evaluate the runtime performance of OccuRE, the execution times were logged while executing the seven building stream configurations listed in Table IV. Figure 8 disects the running time collected while executing each building stream configuration. Each strategy were executed every 30 seconds for 15 minutes. The results demonstrate considering (*Now-PIR, Now-Counts, Now-Fusion, Now-Building, Now-1Hour*) that most time is spend getting data from the sMAP server and very little on strategy resolution and processing. As no model is learned zero time is used on model learning. Comparing *Now-Counts* and *Now-PIR* the faster data loading and processing of *Now-Counts* can be explained by the number of camera count lines and PIR sensors of 30 and 68, respectively. For *Past-PIR* more time is spend on gathering data as data for a longer time period is loaded and therefore also more time is spend on processing the data. For *Future-PIR* time is spend on learning a model and processing data to produce predictions. In all cases only an insignificant amount of time is used for resolving the strategies to apply. On average four of the seven configurations used less than a second to execute, *Now-Fusion* used just above a second, *Future-PIR* used around 21 seconds for model learning and around 3 seconds for prediction and *Past-PIR* used just below 10 seconds. Therefore it can be concluded that the platform could provide a continuous calculation of the occupancy information within a 30 seconds interval for all the spaces in the considered building.

Accuracy of Occupancy Information. The main contribution of OccuRE is building portability and ease of development independently of the accuracy provided by the processing strategies. Therefore, the accuracy evaluation focuses on a subset of the strategies and is included to mainly document the correctness of the information computed by the platform. To evaluate the accuracy we compare the use of PIR and camera count sensors to provide an occupancy count for building M. As ground truth we have collected human observations for the total count of people in building M by monitoring the entries. Figure 9 presents the occupancy counts based on ground truth, OccuRE with PIR sensors and OccuRE with camera count lines. From the figure we can observe that: 1) The camera count lines demonstrate a high accuracy for people counting with a Root Mean Squared Error (RMSE) of 3.3. The only type of error is a tendency to under counting, e.g., when around twenty people enter the building a few

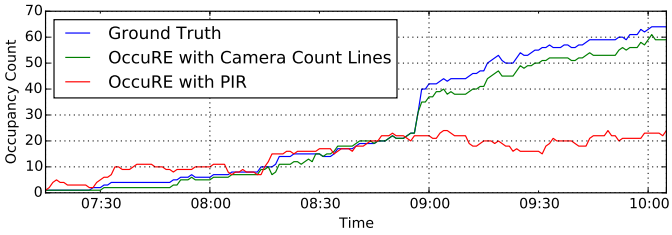

Fig. 9. Accuracy of OccuRE computed occupancy information.

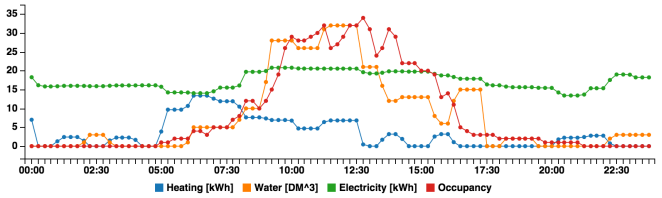

Fig. 10. Diagnostics for Building M highlighting excessive non-occupancy driven loads.

minutes before 9 o'clock not all of them are counted. 2) The PIR sensors provide a low accuracy with a RMSE of 21.7. The error types are over counting when there is a low amount of people in the building that triggers more than one sensor and under counting when many people only trigger one sensor. The results illustrate the different accuracies achievable by OccuRE depending on the available building instrumentation. Applications are provided with the estimated accuracy of information so they can adjust their logic based on the accuracy.

Building Portability. The previous sections covered the use of OccuRE for building M. To evaluate OccuRE's ability to be ported to other buildings we report our experience of porting OccuRE to two other buildings: the GreenTech House in Vejle, Denmark and the Sutardja Dai Hall, Berkeley, US. With this goal we configured the ODataDriver and sMAPWrapper drivers implemented for the purpose listed in Section VI, respectively. The drivers enabled us to stream data from the building instrumentation of the two buildings. The drivers also provided the mapping of metadata and sensor values. In both cases it only took 3-4 hours to implement the drivers and configure the buildings showing the ease of porting OccuRE to other buildings. Furthermore, adding the new drivers did not create any ripple effects on other parts of the OccuRE platform. If we already had the drivers the porting would only had been a matter of configuration. Thereby, we can compute occupancy information for each of the buildings.

Strategy Extensions. As concluded for design goal DG2 a wide range of sensor modalities is relevant for computing occupancy information. As illustrated above drivers can easily be added to integrate with new types of occupancy sensing systems. Therefore, it is important that existing processing strategies can be extended or new strategies added that can handle the new sensor modality. In the following we will describe the actions for adding a strategy that use CO_2 sensors for estimating occupancy counts for the now. The actions needed are as follows: 1) Implement a new strategy based on the interface `OccuREStrategy` that accepts CO_2 readings and return occupancy counts. 2) estimate the quality of the strategy as the coverage of CO_2 sensors. 3) Reuse an existing strategy for the grouping of data points (`BySpatialGranularity`) as there is normally one sensor per space. 4) Reuse a selection of data for the now (`DataSelectionNow`). 5) Implement a method for estimating occupancy counts from CO_2 readings. The simplest method is to estimate occupancy counts assuming occupants as the only CO_2 source. However, this method will be very inaccurate if a ventilation system is present which calls for applying more advanced models [28], [30] that can take a ventilation system into account. As we already have two building instrumentation drivers that can collect readings from CO_2 sensors with this strategy in place we are, for instance, ready to apply the methods in the GreenTech House and the Sutardja Dai Hall.

VIII. Prototype Applications

In this section we report on our experiences with building a number of prototype applications using the API provided by OccuRE. The purpose is to demonstrate that the API of OccuRE enables relevant occupancy-driven applications and that by using the OccuRE API the applications become highly portable with regards to occupancy information. We also compare the situation of developing the prototypes with and without OccuRE. For the comparison we state the number of code lines saved in the prototype implementation by using OccuRE compared to developing the prototypes with sMAP only.

Building Diagnostics. Occupant behavior is important to consider when diagnosing the energy consumption of a building. To evaluate this case a building diagnostic tool was extended to include data from OccuRE computed via the configuration listed in Table VI. Figure 10 shows the electricity, heat and water consumption data for building M over 24 hours together with occupancy presence as counts computed by OccuRE. The data provides a premise to analyze the correlations between occupancy and consumption. From the figure obtained it is clear that equipment in the building need to be reconfigured or replaced to minimize loads when there is no occupants in the building. For our implementation the total lines of code avoided in application code by using OccuRE is 266.

Occupancy-driven Ventilation. Controlling ventilation systems in response to occupancy change is crucial for improving the overall energy efficiency of ventilation systems. One example is demand controlled ventilation that controls the air flow rate with regards to estimated occupancy [31]. We implemented a setpoint scheduler that can change ventilation setpoints following the occupancy presence count computed by OccuRE using the configuration listed in Table VI. For our implementation the total lines of code avoided in application code by using OccuRE is 269.

Occupant-driven Heating. Superfluous heating can be avoided if systems are able to heat rooms to a comfort temperature when occupied. As room heating is not instantaneous to achieve this goal requires the ability to predict when rooms are

	Building Diagnostics	Occupancy-driven Ventilation	Occupant-driven Heating
temporalcoverage	past 1days	now	future 1days
temporalgranularity	15min	available	5min
spatialcoverage	["Metadata/Building='M'"]	["Metadata/Space='0.34'"]	["Metadata/Space='1.28'"]
spatialgranularity	Metadata/Building	Metadata/Space	Metadata/Space
informationtype	presence-count	presence-count	presence-boolean
Chosen strategy	Sum count line crossing, Past	Sum count line crossing, Now	Boolean operator on motion events, Future

TABLE VI
BUILDING CONFIGURATIONS FOR THREE DIFFERENT APPLICATIONS.

occupied. We implemented the pseudo-code given in the top of Figure 4 to schedule setpoints for a Z-wave controlled Danfoss Living Connect thermostat. The value of *timetooccupancy* is updated from a stream generated by OccuRE with the building configuration listed in Table VI. For our implementation the total lines of code avoided in application code by using OccuRE is 294.

Summary of Results. The cases demonstrate that it was feasible to use the OccuRE API to supply the data requested by the prototype applications. The implementations were concise because the applications only had to request data from the platform and specify the building stream configuration. In all cases more than 250 lines of rather complex sensor data processing code was avoided easing the development of applications. The applications in all cases improved their building portability as they were decoupled from the particular building instrumentation with regards to occupancy information. Comparing the chosen strategies in the three cases we can see that OccuRE adapted to the available building instrumentation. For building diagnostics the OccuRE platform utilized the Xovis camera counters installed at all entries of the building to calculate the total building occupancy count. This strategy returns the highest quality estimates compared to other strategies, e.g., based on PIR data. Also for the occupancy-driven ventilation the platform selected to use a strategy using camera count lines from two Xovis camera counters installed at the two entrances to the room. For occupancy-driven heating OccuRE selected a strategy for predicting occupancy based on PIR data as in the particular room only a PIR sensor was available.

IX. LIMITATIONS

The limitations of the presented work concern the considered size of buildings, breadth of sensor modalities and duration of deployment studies. In this work we evaluated the platform for buildings with up to 102 sensors per building without any performance restrictions, however, it would interesting to study scalability to large building complexes. We considered six types of occupancy sensor modalities representing very different modalities but it would be interesting to study the consequences if any of adding additional sensor modalities. Finally, it would be relevant to evaluate deployment studies with a wider selection of applications.

X. RELATED WORK

In this section we cover related work in terms of occupancy sensing systems, BMS and BOS, context-awareness and component platforms. In addition to this a number of general platforms are available for processing of time-series data from sensors. This work differ in comparison to such general platforms by providing the domain-specialized abstractions and processing strategies to ease the development of occupancy-driven applications.

Occupancy Sensing Systems. Work on occupancy sensing systems have primarily focused on sensing methods and sensor design [8], [9], [10], [11]. This trend has resulted in a range of solutions with technology-specific APIs that statically couple individual sensors to the type of computed occupancy information as highlighed earlier. Therefore, there is a need for a platform with an API that provide technology agnostic abstractions.

BMS and BOS. Commercial available Building Management Systems (BMS) as surveyed by Granderson et al. [32] include capabilities to react to Boolean information about occupancy presence in relevant spaces. However, the systems do not provide good APIs for development of occupancy-driven applications including advanced algorithms for fusion and prediction of occupancy behavior. A BOS compared to a classic BMS provides APIs for the development of sandboxed applications for buildings. Existing examples of BOS platforms provide limited constructs for directly supporting occupancy-driven applications. For instance, sMAP [4] provides some constructs for handling the temporal granularity of sensor streams, and BuildingDepot [5]) and BOSS [3] provide concepts for describing the spatial placement of the building instrumentation. Open source community projects (e.g. OpenHAB [6]) and industry driven efforts (e.g., HomeOS [7]) do not include any constructs to directly support occupancy-driven building applications beyond reading raw sensor values.

Context-Awareness. The area of context-aware computing is trying to improve the computing experience by utilizing context information. Examples of software platforms include the Context Toolkit [33] and the component model by Grisworld et al. [34] both proposed to support the development of context-aware applications. The main difference between context-aware applications and occupancy-driven applications is that the former focus on devices and individuals with known identities where as the later primarily focus on physical spaces and people with often unknown identities. Therefore, these platforms can act as inspiration but do not provide the needed constructs for occupancy-driven applications.

Component Platforms for Smart Buildings. Cid et al. [35] propose a component model for sensing applications in smart offices. They target the dynamic reconfiguration of applications that run partly on wireless sensing nodes. Druilhe et al. [36] propose a component model for digital homes designed to run applications on the fewest nodes in a home to

improve energy efficiency. OccuRE has a different scope from such efforts but is inspired by such work with regards to the chosen design for the computation model.

XI. Conclusions

In this paper we presented the design, implementation and evaluation of OccuRE, a stream-based occupancy reasoning platform. The platform extends BOS platform functionality with a component-based computation model for occupancy information and a technology agnostic API for occupancy information. These extensions enable the development of portable occupancy-driven applications and ease application development. Through micro-benchmarks we showed that OccuRE successfully, efficiently and in a portable manner computes a variety of occupancy information given the technology-heterogeneous building instrumentation of three different buildings. We developed three prototype applications in order to evaluate our platform and demonstrate that the API of OccuRE enables several types of occupancy-driven applications, that the applications by using the interface achieve portability in regards to occupancy information computation and that the platform eases the development of occupancy-driven applications. Thereby, the OccuRE platform is an important step towards the development of portable occupancy-driven applications to improve the energy performance of buildings.

Acknowledgment

This work is supported by the Innovation Fund Denmark for the project COORDICY (4106-00003B).

References

[1] T. A. Nguyen and M. Aiello, "Energy intelligent buildings based on user activity: A survey," *Energy and Buildings*, vol. 56, no. 0, pp. 244 – 257, 2013.

[2] A. Caucheteux, A. Es Sabar, and V. Boucher, "Occupancy measurement in building: A litterature review, application on an energy efficiency research demonstrated building," *International Journal of Metrology and Quality Engineering*, vol. 4, pp. 135–144, 1 2013.

[3] S. Dawson-Haggerty, A. Krioukov, J. Taneja, S. Karandikar, G. Fierro, N. Kitaev, and D. E. Culler, "BOSS: building operating system services," in *USENIX NSDI 2013*, 2013, pp. 443–457.

[4] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. E. Culler, "smap: a simple measurement and actuation profile for physical information," in *SenSys 2010*, 2010, pp. 197–210.

[5] T. Weng, A. Nwokafor, and Y. Agarwal, "Buildingdepot 2.0: An integrated management system for building analysis and control," in *BuildSys 2013*, 2013, pp. 7:1–7:8.

[6] "Openhab: http://www.openhab.org."

[7] "Homeos: http://research.microsoft.com/en-us/projects/homeos/."

[8] K. Christensen, R. Melfi, B. Nordman, B. Rosenblum, and R. Viera, "Using existing network infrastructure to estimate building occupancy and control plugged-in devices in user workspaces," *Int. J. Commun. Netw. Distrib. Syst.*, vol. 12, no. 1, pp. 4–29, Nov. 2014.

[9] Y. Agarwal, B. Balaji, R. E. Gupta, J. Lyles, M. Wei, and T. Weng, "Occupancy-driven energy management for smart building automation," in *BuildSys'10*, 2010, pp. 1–6.

[10] V. Erickson, M. A. Carreira-Perpinan, and A. E. Cerpa, "OBSERVE: Occupancy-based system for efficient reduction of HVAC energy," in *IPSN 2011*, 2011, pp. 258–269.

[11] J. Scott, A. J. B. Brush, J. Krumm, B. Meyers, M. Hazas, S. Hodges, and N. Villar, "Preheat: controlling home heating using occupancy prediction," in *UbiComp 2011*, 2011, pp. 281–290.

[12] R. Melfi, B. Rosenblum, B. Nordman, and K. Christensen, "Measuring building occupancy using existing network infrastructure," in *IGCC 2012*, 2011, pp. 1–8.

[13] R. Attar, E. Hailemariam, S. Breslav, A. Khan, and G. Kurtenbach, "Sensor-enabled cubicles for occupant-centric capture of building performance data." *ASHRAE Transactions*, vol. 117, no. 2, 2011.

[14] J. Zhao, B. Lasternas, K. P. Lam, R. Yun, and V. Loftness, "Occupant behavior and schedule modeling for building energy simulation through office appliance power consumption data mining," *Energy and Buildings*, vol. 82, pp. 341–355, 2014.

[15] W.-K. Chang and T. Hong, "Statistical analysis and modeling of occupancy patterns in open-plan offices using measured lighting-switch data," *Building Simulation*, 01/2013 2013.

[16] "Revealing occupancy patterns in an office building through the use of occupancy sensor data," *Energy and Buildings*, vol. 67, no. 0, pp. 587 – 595, 2013.

[17] R. K. Harle and A. Hopper, "The potential for location-aware power management," in *UbiComp 2008*, 2008, pp. 302–311.

[18] M. Saha, S. Thakur, A. Singh, and Y. Agarwal, "Energylens: combining smartphones with electricity meter for accurate activity detection and user annotation," in *e-Energy '14*, 2014, pp. 289–300.

[19] A. Krioukov and D. Culler, "Personal building controls," in *IPSN 12*, 2012, pp. 157–158.

[20] C. Koehler, B. D. Ziebart, J. Mankoff, and A. K. Dey, "Therml: occupancy prediction for thermostat control," in *UbiComp '13*, 2013, pp. 103–112.

[21] E. Patti, A. Acquaviva, M. Jahn, F. Pramudianto, R. Tomasi, D. Rabourdin, J. Virgone, E. Macii *et al.*, "event-driven user-centric middleware for energy-efficient buildings and public spaces," *Systems Journal, IEEE*, vol. 99, pp. 1–10, 2014.

[22] B. Balaji, J. Xu, A. Nwokafor, R. Gupta, and Y. Agarwal, "Sentinel: occupancy based HVAC actuation using existing wifi infrastructure within commercial buildings," in *SenSys'13*, 2013, p. 17.

[23] T. A. Nguyen and M. Aiello, "Beyond indoor presence monitoring with simple sensors," in *PECCS 2012*, 2012, pp. 5–14.

[24] I. Georgievski, T. A. Nguyen, and M. Aiello, "Combining activity recognition and AI planning for energy-saving offices," in *IEEE UIC*, 2013, pp. 238–245.

[25] A. Beltran, V. L. Erickson, and A. E. Cerpa, "Thermosense: Occupancy thermal based sensing for hvac control," in *BuildSys'13*, 2013, pp. 11:1–11:8.

[26] V. L. Erickson, Y. Lin, A. Kamthe, R. Brahme, A. Surana, A. E. Cerpa, M. D. Sohn, and S. Narayanan, "Energy efficient building environment control strategies using real-time occupancy measurements," in *BuildSys '09*, 2009, pp. 19–24.

[27] B. Dong and K. P. Lam, "Building energy and comfort management through occupant behaviour pattern detection based on a large-scale environmental sensor network," *Journal of Building Performance Simulation*, vol. 4, no. 4, pp. 359–369, 2011.

[28] A. Ebadat, G. Bottegal, D. Varagnolo, B. Wahlberg, and K. H. Johansson, "Estimation of building occupancy levels through environmental signals deconvolution," in *BuildSys 2013*, 2013, pp. 1–8.

[29] K. Imamovic and M. B. K. Fisayo Caleb Sangogboye, "Poster abstract: Improving occupancy presence prediction via multi-label classification," in *BuildSys 2015*, 2015.

[30] W. S and J. X, "Co2-based occupancy detection for on-line outdoor air flow control," *Indoor Built Environment*, vol. 7, pp. 165–181, 1998.

[31] T. Lawrence and J. Braun, "Calibrated simulation for retrofit evaluation of demand-controlled ventilation in small commercial buildings," *ASHRAE Trans*, pp. 227–240, 2007.

[32] J. Granderson, M. Piette, G. Ghatikar, and P. Price, "Building energy information systems: State of the technology and user case studies," November 2009.

[33] D. Salber, A. K. Dey, and G. D. Abowd, "The context toolkit: Aiding the development of context-enabled applications," in *CHI*, 1999, pp. 434–441.

[34] W. G. Griswold, R. T. Boyer, S. W. Brown, and T. M. Truong, "A component architecture for an extensible, highly integrated context-aware computing infrastructure," in *ICSE*, 2003, pp. 363–373.

[35] P. J. del Cid, D. Hughes, S. Michiels, and W. Joosen, "Ensuring application integrity in shared sensing environments," in *CBSE*, 2014, pp. 149–158.

[36] R. Druilhe, M. Anne, J. Pulou, L. Duchien, and L. Seinturier, "Components mobility for energy efficiency of digital home," in *CBSE*, 2013, pp. 153–158.